
EEE146
STRINGS

What is a String?

- A string is actually a character array.
  - You can use it like a regular array of characters.
  - However, it has also some unique features that make string processing easy.

What You Should Keep in Mind

1. Anything written in double quotes (" ") is a string.
2. The end of a string is always marked with the invisible null character '\0'.
   - We say "a string is always null-terminated."
   - All string functions depend on this null character. If you ignore the null character, everything will fail.
   - You have to take into account that null character also consumes one byte.

Strings

- So, the idea is simple: Obey the following simple rules:
  - When you define the string variable, don't forget to add one byte for the null character.
  - All string functions depend on the null character so don't mess around with the null character.
    - Anything after the null character will be ignored.
    - Anything up to the null character will be considered.

Strings

There are multiple string functions that help programming. Learn what type of arguments are required and what they return.

Initializing Strings

- Instead of initializing the elements of a string one-by-one, you can initialize it using a string.

Eg:

```
string st;
st="Gaziantep University";
```

Eg:

```
char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};
char greeting[] = "Hello";
```

Input string with getline()

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
    string st;
    getline(cin,st);
    cout<<st<<endl;
    return 0;
}
```

getline() function

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
  char mybuffer [100];
  cout << "What's your name? ";
  cin.getline (mybuffer,100);
  cout << "Hello " << mybuffer << ".\n";
  cout << "Which is your favourite team? ";
  cin.getline (mybuffer,100);
  cout << "I like " << mybuffer << " too.\n";
  return 0;
}
```

String Functions

.length()

.size()

.clear()

.erase()

.copy()

.insert()

.append()

.compare()

.length()

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
  string str ("Test string");
  cout << "The size of str is " << str.length() << "
  bytes.\n";
  return 0;
}
```

.size()

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
  string str ("Test string");
  cout << "The size of str is " << str.size() << "
  bytes.\n";
  return 0;
}
```

.clear()

```
#include <iostream>
#include <string>
using namespace std;
int main()
{   char c; string str;
    cout << "Please type some lines of text. Enter a dot (.)
   to finish:\n";
    do {
     c = cin.get();
     str += c;
     if (c=='\n')
     {   cout << str;
         str.clear();}
    } while (c!='.');
    return 0;
}
```

.erase()

std::string::erase

<string>

C++98 | C++11

```
sequence (1)    string& erase (size_t pos = 0, size_t len = npos);
character (2)   iterator erase (iterator p);
range (3)       iterator erase (iterator first, iterator last);
```

Erase characters from string

Erases part of the string, reducing its length:

(1) sequence

Erases the portion of the string value that begins at the character position *pos* and spans *len* characters (or until the *end of the string*, if either the content is too short or if *len* is string::npos.
Notice that the default argument erases all characters in the string (like member function clear).

(2) character

Erases the character pointed by *p*.

(3) range

Erases the sequence of characters in the range [first,last).

Parameters

`pos`

Position of the first character to be erased.
If this is greater than the string length, it throws out_of_range.
Note: The first character in *str* is denoted by a value of `0` (not `1`).

`len`

Number of characters to erase (if the string is shorter, as many characters as possible are erased).
A value of string::npos indicates all characters until the end of the string.

`p`

Iterator to the character to be removed.

`first, last`

Iterators specifying a range within the string] to be removed: [first,last). i.e., the range includes all the characters between *first* and *last*, including the character pointed by *first* but not the one pointed by *last*.

`size_t` is an unsigned integral type (the same as member type `string::size_type`).
Member types `iterator` and `const_iterator` are random access iterator types that point to characters of the string.

Return value

The *sequence version (1)* returns `*this`.
The others return an iterator referring to the character that now occupies the position of the first character erased, or string::end if no such character exists.

Member type `iterator` is a random access iterator type that points to characters of the string.

.erase()

```
#include <iostream>
#include <string>

int main ()
{
  std::string str ("This is an example sentence.");
  std::cout << str << '\n';
                                          // "This is an example sentence."
  str.erase (10,8);                       //            ^^^^^^^^
  std::cout << str << '\n';
                                          // "This is an sentence."
  str.erase (str.begin()+9);              //           ^
  std::cout << str << '\n';
                                             // "This is a sentence."
  str.erase (str.begin()+5, str.end()-9);  //       ^^^^^
  std::cout << str << '\n';
                                             // "This sentence."
  return 0;
}
```

```
This is an example sentence.
This is an sentence.
This is a sentence.
This sentence.
```

.copy()

```
//size_t copy (char* s, size_t len, size_t pos = 0) const;
```

std::string::copy

<string>

```
size_t copy (char* s, size_t len, size_t pos = 0) const;
```

Copy sequence of characters from string

Copies a substring of the current value of the string object into the array pointed by *s*. This substring contains the *len* characters that start at position *pos*.

The function does not append a null character at the end of the copied content.

Parameters

`s`

Pointer to an array of characters.
The array shall contain enough storage for the copied characters.

`len`

Number of characters to copy (if the string is shorter, as many characters as possible are copied).

`pos`

Position of the first character to be copied.
If this is greater than the string length, it throws out_of_range.
Note: The first character in the string is denoted by a value of `0` (not `1`).

Return value

The number of characters copied to the array pointed by *s*. This may be equal to *len* or to `length()-pos` (if the string value is shorter than `pos+len`).

`size_t` is an unsigned integral type (the same as member type `string::size_type`).

.copy()

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
  char buffer[20];
  string str="Test string...";
  unsigned int length;
  length = str.copy(buffer,6,5);
  buffer[length]='\0';
  cout << "buffer contains: " << buffer << '\n';
  return 0;
}
```

buffer contains: string

.insert()

std::string::insert

<string>

C++98 | C++11 | C++14

```
string (1)          string& insert (size_t pos, const string& str);
substring (2)       string& insert (size_t pos, const string& str, size_t subpos, size_t sublen);
c-string (3)        string& insert (size_t pos, const char* s);
buffer (4)          string& insert (size_t pos, const char* s, size_t n);
fill (5)            string& insert (size_t pos, size_t n, char c);
                      void insert (iterator p, size_t n, char c);
single character (6) iterator insert (iterator p, char c);
range (7)           template <class InputIterator>
                      void insert (iterator p, InputIterator first, InputIterator last);
```

Insert into string

Inserts additional characters into the string right before the character indicated by *pos* (or *p*):

Parameters

`pos`
Insertion point: The new contents are inserted before the character at position *pos*.
If this is greater than the object's length, it throws out_of_range.
Note: The first character is denoted by a value of `0` (not `1`).

`str`
Another string object.

`subpos`
Position of the first character in *str* that is inserted into the object as a substring.
If this is greater than *str*'s length, it throws out_of_range.
Note: The first character in *str* is denoted by a value of `0` (not `1`).

`sublen`
Length of the substring to be copied (if the string is shorter, as many characters as possible are copied).
A value of npos indicates all characters until the end of *str*.

`s`
Pointer to an array of characters (such as a *c-string*).

`n`
Number of characters to insert.

`c`
Character value.

`p`
Iterator pointing to the insertion point: The new contents are inserted before the character pointed by *p*.
`iterator` is a member type, defined as a random access iterator type that points to characters of the string.

`first, last`
Input iterators to the initial and final positions in a range. The range used is `[first,last)`, which includes all the characters between *first* and *last*, including the character pointed by *first* but not the character pointed by *last*.
The function template argument `InputIterator` shall be an input iterator type that points to elements of a type convertible to `char`.

`il`
An initializer_list object.
These objects are automatically constructed from *initializer list* declarators.

.insert()

```
#include <iostream>
#include <string>
using namespace std;
int main()
{  string str="to be question";
  string str2="the ";
  string str3="or not to be";
  str.insert(6,str2);
  str.insert(6,str3,3,4);
  str.insert(10,"that is cool",8);
  str.insert(10,"to be ");
  str.insert(15,1,':');
  str.insert(str.begin()+5,',');
  str.insert (str.end(),3,'.');
  str.insert (str.begin()+6,str3.begin(),str3.begin()+3);
  cout << str << '\n';
  return 0;
}
```

```
to be the question
to be not the question
to be not that is the question
to be not to be that is the question
to be not to be: that is the question
to be, not to be: that is the question
to be, not to be: that is the question...
to be,or  not to be: that is the question...
```

.append()

std::string::append

<string>

C++98 | C++11 | C++14

string (1)	`string& append (const string& str);`
substring (2)	`string& append (const string& str, size_t subpos, size_t sublen);`
c-string (3)	`string& append (const char* s);`
buffer (4)	`string& append (const char* s, size_t n);`
fill (5)	`string& append (size_t n, char c);`
range (6)	`template <class InputIterator>` `  string& append (InputIterator first, InputIterator last);`

Append to string

Extends the string by appending additional characters at the end of its current value:

Parameters

`str`
Another string object, whose value is appended.

`subpos`
Position of the first character in *str* that is copied to the object as a substring.
If this is greater than *str*'s length, it throws out_of_range.
Note: The first character in *str* is denoted by a value of `0` (not `1`).

`sublen`
Length of the substring to be copied (if the string is shorter, as many characters as possible are copied).
A value of string::npos indicates all characters until the end of *str*.

`s`
Pointer to an array of characters (such as a *c-string*).

`n`
Number of characters to copy.

`c`
Character value, repeated *n* times.

`first, last`
Input iterators to the initial and final positions in a range. The range used is `[first,last)`, which includes all the characters between *first* and *last*, including the character pointed by *first* but not the character pointed by *last*.
The function template argument `InputIterator` shall be an input iterator type that points to elements of a type convertible to `char`.
If `InputIterator` is an integral type, the arguments are casted to the proper types so that signature (5) is used instead.

`il`
An initializer_list object.
These objects are automatically constructed from *initializer list* declarators.

size_t is an unsigned integral type.

.append()

```
#include <iostream>
#include <string>
using namespace std;
int main()
{ string str;
  string str2="Writing ";
  string str3="print 10 and then 5 more";
  str.append(str2);
  str.append(str3,6,3);
  str.append("dots are cool",5);
  str.append("here: ");
  str.append(10,'.');
  str.append(str3.begin()+8,str3.end());
  str.append<int>(5,0x2E);
  cout << str << '\n';
  return 0;
}
```

```
Writing
Writing 10
Writing 10 dots
Writing 10 dots here:
Writing 10 dots here: ..........
Writing 10 dots here: .......... and then 5 more
Writing 10 dots here: .......... and then 5 more.....
```

.compare()

std::string::compare

<string>

C++98 | C++11 | C++14

```
string (1)      int compare (const string& str) const;
substrings (2)  int compare (size_t pos, size_t len, const string& str) const;
                int compare (size_t pos, size_t len, const string& str,
                             size_t subpos, size_t sublen) const;
c-string (3)    int compare (const char* s) const;
                int compare (size_t pos, size_t len, const char* s) const;
buffer (4)      int compare (size_t pos, size_t len, const char* s, size_t n) const;
```

Compare strings

Compares the value of the string object (or a substring) to the sequence of characters specified by its arguments.

The *compared string* is the value of the string object or -if the signature used has a *pos* and a *len* parameters- the substring that begins at its character in position *pos* and spans *len* characters.

This string is compared to a *comparing string*, which is determined by the other arguments passed to the function.

Parameters

`str`
Another string object, used entirely (or partially) as the *comparing string*.

`pos`
Position of the first character in the *compared string*.
If this is greater than the string length, it throws out_of_range.
Note: The first character is denoted by a value of `0` (not `1`).

`len`
Length of *compared string* (if the string is shorter, as many characters as possible).
A value of string::npos indicates all characters until the end of the string.

`subpos, sublen`
Same as *pos* and *len* above, but for the *comparing string*.

`s`
Pointer to an array of characters.
If argument *n* is specified *(4)*, the first *n* characters in the array are used as the *comparing string*.
Otherwise *(3)*, a null-terminated sequence is expected: the length of the sequence with the characters to use as *comparing string* is determined by the first occurrence of a null character.

`n`
Number of characters to compare.

.compare()

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
  string str1 ("green apple");
  string str2 ("red apple");
  if (str1.compare(str2) != 0)
    cout << str1 << " is not " << str2 << '\n';
  if (str1.compare(6,5,"apple") == 0)
    cout << "still, " << str1 << " is an apple\n";
  if (str2.compare(str2.size()-5,5,"apple") == 0)
    cout << "and " << str2 << " is also an apple\n";
  if (str1.compare(6,5,str2,4,5) == 0)
    cout << "therefore, both are apples\n";
  return 0;
}
```

```
green apple is not red apple
still, green apple is an apple
and red apple is also an apple
therefore, both are apples
```